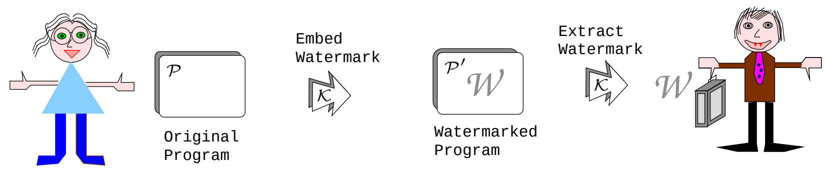
Software Watermarking Through Register Allocation: Implementation, Analysis, and Attacks

Ginger Myles
Christian Collberg
{mylesg,collberg}@cs.arizona.edu

University of Arizona
Department of Computer Science

What is Software Watermarking?

- Technique used to aid in the prevention of software piracy.
- `embed`$(P, w, key) \rightarrow P'$
- `recognize`$(P', key) \rightarrow w$
- Watermark: w uniquely identifies the owner of P.
- Fingerprint: w uniquely identifies the purchaser of P.



What is Software Watermarking?

- Static: the watermark is stored directly in the data or code sections of a native executable or class file. Make use of the features of an application that are available at compile-time.
- Dynamic: the watermark is stored in the run-time structures of the program.

What is Software Watermarking?

- Blind: the recognizer is given the watermarked program and the watermark key as input.
- Informed: the recognizer is given the watermarked program and the watermark key as input and it also has access to the unwatermarked program.

Why use Software Watermarking?

- Discourages illegal copying and redistribution.
- A copyright notice can be used to provide proof of ownership.
- A fingerprint can be used to trace the source of the illegal redistribution.
- Does not prevent illegal copying and redistribution.

How can we watermark software?

- Insert new (non-functional or nonexcuted) code
- Reorder code where it does not change the functionality
- Manipulate instruction frequencies

```
switch (E) {
   case 1: {···}
   case 5: {···}
   case 9: {···}
}
```

$\Longrightarrow$

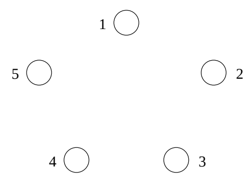
```
switch (E) {
   case 5: {···}
   case 1: {···}
   case 9: {···}
}
```


Interference Graph

- Models the relationship between the variables in the procedure.
- Each variable in the procedure is represented by a vertex.
- If two variables have overlapping live ranges then the vertices are joined by an edge.
- Color the graph such that
  - minimize the number of registers required
  - two live variables do not share a register

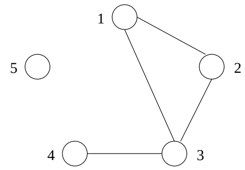
Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```



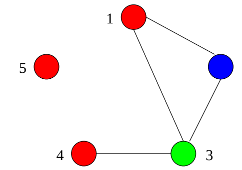
Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```



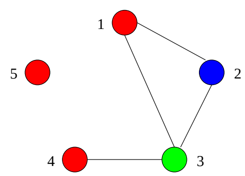
Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```



Interference Graph Example

```
v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
```



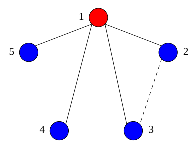

```
mult R1, 2, 2
mult R2, 2, 3
mult R3, 2, R2
add  R1, R1, R2
mult R1, 3, R3
```

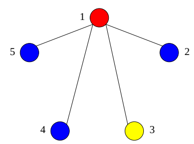

QP Algorithm

- Originally proposed by G. Qu and M. Potkonjak.
- *Constraint-based* software watermarking algorithm.
- Embeds a watermark in the register allocation of the program through a technique called *edge-adding*.
  - Use the interference graph and the graph coloring problem to embed a watermark in the register allocation.

QP Algorithm

- Edges are added between chosen vertices in a graph based on the value of the message.
- Since the vertices are now connected, they cannot be assigned to the same register.



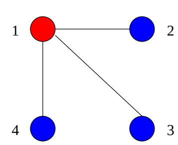


QPS Algorithm

- Improvement on the QP Algorithm
- In order to use the algorithm for software watermarking, stricter embedding criteria are required.
  - Unpredicability of the coloring of vertices using the original algorithm.
  - One vertex could be used multiple times.

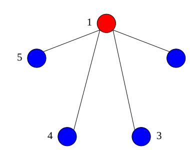
QPS Algorithm

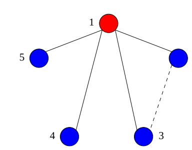
- Key idea:
  - Select triples of vertices such that they are isolated units that will not effect other vertices in the graph.
  - Colored Triple: Given an n-colorable graph $G = (V, E)$, a set of three vertices $\{v_1, v_2, v_3\}$ is considered a colored triple if
    1. $v_1, v_2, v_3 \in V$,
    2. $(v_1, v_2), (v_1, v_3), (v_2, v_3) \notin E$, and
    3. v_1, v_2, v_3 are all colored the same color.

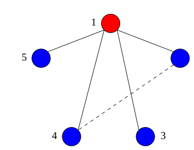


QPS Embedding Algorithm

for each vertex $v_i \in V$ which is not already in a triple
    if possible find the nearest two vertices v_{i_1} and v_{i_2}
    such that
        v_{i_1} and v_{i_2} are the same color as v_i,
        and v_{i_1} and v_{i_2} are not already in triple.
    if $m_i = 0$
        add edge (v_i, v_{i_1})
    else
        add edge (v_i, v_{i_2})
end for

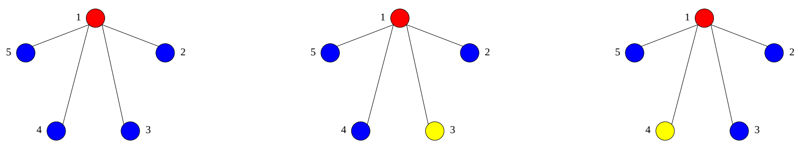




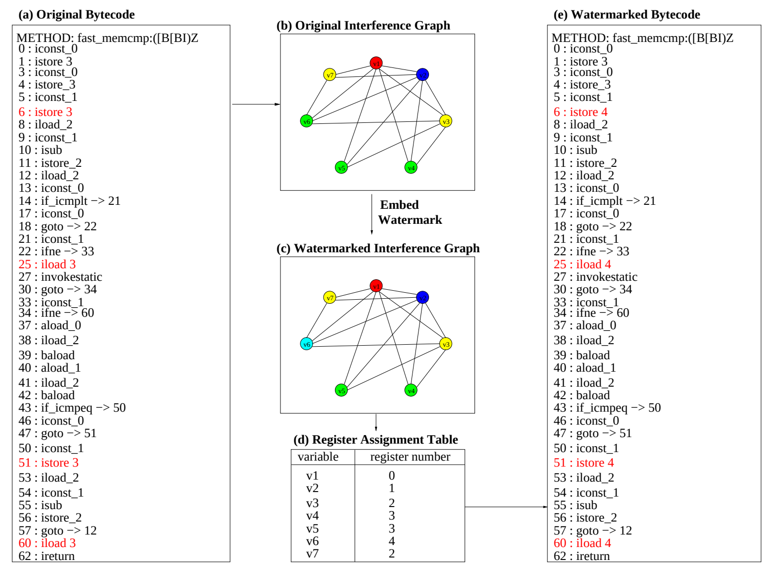


QPS Recoginition Algorithm

for each vertex $v_i \in V$ which is not already in a triple
    if possible find the nearest two
        vertices v_{i_1} and v_{i_2} such that
        v_{i_1} and v_{i_2} are the same color as v_i,
        and v_{i_1} and v_{i_2} are not already in triple.
    if v'_i and v'_{i_1} are different colors
        found a 0
        add edge (v_i, v_{i_1})
    else
        found a 1
        add edge (v_i, v_{i_2})
end for



QPS Example



Implementation

- Implemented in Java using the BCEL bytecode editor.
- Incorporated into the SandMark framework.
- Can be applied to an entire application or a single class file.
- Watermark is spread across all classes and is embedded as many times as possible.
- Message is converted to a binary representation using the ASCII value of the characters. An 8-bit length field is added to the beginning that is used in the recognition phase.

QPS Watermarking Algorithm Evaluation

- Performed a variety of empirical tests to evaluate the algorithm's overall effectiveness.
- Implementation within SandMark facilitated the study of manual attacks and the application of obfuscations.
- The evaluation examined five software watermarking properties.

Watermark Evaluation Properties

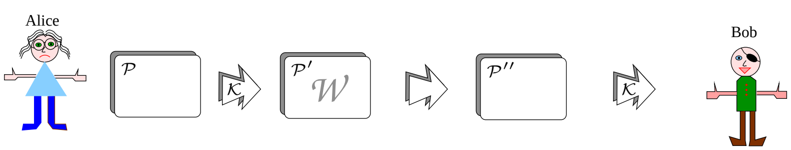
- Credibility: The watermark should be readily detectable for proof of authorship while minimizing the probability of coincidence.
- Data-rate: Maximize the length of message that can be embedded.
- Perceptual Invisibility (Stealth): A watermark should exhibit the same properties as the code around it so as to make detection difficult.
- Part Protection: A good watermark should be distributed throughout the software in order to protect all parts of it.

Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks

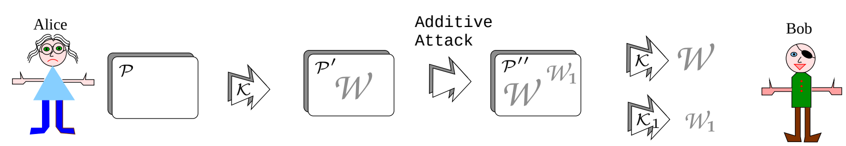
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Subtractive Attack: The adversary attempts to remove all or part of the watermark.



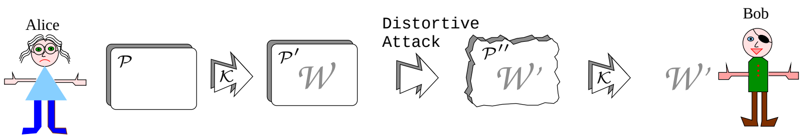
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Additive Attack: The adversary adds a new watermark.



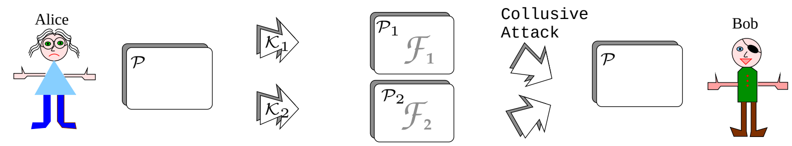
Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Distortive Attack: The attacker applies a series of semantics-preserving transformations to render the watermark useless.



Watermark Evaluation Properties

- Resilience: A watermark should withstand a variety of attacks
  - Collusive Attack: The adversary compares two differently fingerprinted copies of the software to identify the location.



Results

- The original QP algorithm has a very low recovery rate.
- Accurate recovery is a necessity so the QPS algorithm was developed.
- The QPS algorithm has a very low data-rate.
- The QPS algorithm is susceptible to a variety of simple attacks.
- The QPS algorithm is quite stealthy.
- The QPS algorithm is extremely credible.

A shameless plug to conclude

- Sandmark contains an implementation of the QPS algorithm as well as several other watermarking algorithms
- http://www.cs.arizona.edu/sandmark